U

SECUR

AD-A203 332

IENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. RE Unclassified	1b. RESTRICTIVE MARKINGS DTIC FILE COPY
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S) AFWAL-TR-88-1142
6a. NAME OF PERFORMING ORGANIZATION AFWAL Avionics Laboratory 6b. OFFICE SYMBOL (If applicable) AFWAL/AAAI-3	7a. NAME OF MONITORING ORGANIZATION Information Transmission Branch Radio Systems Group
6c. ADDRESS (City, State, and ZIP Code) AFWAL/AAAI-3 Wright-Patterson AFB OH 45433-6543	7b. ADDRESS (City, State, and ZIP Code) AFWAL/AAAI-3 Wright-Patterson AFB OH 45433-6543
8a. NAME OF FUNDING / SPONSORING ORGANIZATION Air Force Avionics Lab 8b. OFFICE SYMBOL (If applicable) AFWAL/AAAI-3	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code) AFWAL/AAAI-3 Wright-Patterson AFB OH 45433-6543	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. 62204F; PROJECT NO. 2538; TASK NO 02; WORK UNIT ACCESSION NO. 04

11. TITLE (Include Security Classification)

Software Development Guidelines

12. PERSONAL AUTHOR(S)

Ms Denice S Jacobs

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Journal Article	FROM _____ TO _____	88/4/1	4

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	CCB, ICDs, Software Development Cycle, System I&T
17	02	01	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Due to the growing complexity of avionic systems, the development cycle for mission critical software has evolved into a collective process of organized tasks. These tasks are distinct levels of effort which are implemented by the developer to ensure the creation of a reliable, operational system. This paper summarizes four principle tasks which have proven to be excellent procedures for developing avioncs software.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS	UNCLASSIFED

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code)	22c. OFFICE SYMBOL
S. Jacobs	(513) 255-4666	AFWAL/AAAI-3

73, JUN 86 Previous editions are obsolete. SECURITY CLASSIFICATION OF THIS PAGE

(1)

SOFTWARE DEVELOPMENT GUIDELINES

By

Denice S. Jacobs
Aeronautical Systems Division
Air Force Wright Aeronautical Laboratories
Wright-Patterson AFB OH 45433-6543

SUMMARY

Due to the growing complexity of avionic systems, the development cycle for mission critical software has evolved into a collective process of organized tasks. These tasks are distinct levels of effort which are implemented by the developer to ensure the creation of a reliable, operational system. This paper summarizes four principle tasks which have proven to be excellent procedures for developing avionics software. The first and foremost task of the project manager is to establish a Configuration Control Board (CCB) as the central core of technical management. It consists of a group of key hardware and software engineers who mutually govern the status of system development, and incorporate design changes on an agreed-to basis. The second task is to logically separate the software project into well-defined phases of development. This, too, requires the cooperation of both hardware and software teams to work together in accordance with a master schedule. The third task is to create an automated data base which contains the latest interface specifications (ICDs) and system message definitions for use by the engineers. Finally, the last task is to procure hardware emulators and stand-alone test stations as an effective means of testing software prior to system integration and test (I&T).

INTRODUCTION

Due to the growing complexity of avionic systems, the development cycle for mission critical software has become more involved in terms of the difficulty and number of software, interface, and test requirements to be defined. Consequently, the probability of incurring design/coding errors is increased due to written and verbal miscommunications between the hardware and software development groups, information latencies due to the number of developers involved, and inadequate

software testing prior to system I&T. In order to help minimize the aforementioned problems, it was necessary to create several guidelines which 1) enforce a formal communication network between the hardware and software groups; 2) enhance the visibility of software progress; 3) improve information access; and 4) enhance the software verification process. These guidelines evolved into four specific tasks which are addressed in the following paragraphs.

TASK 1: CCB MANAGEMENT

The first task is to establish a Configuration Control Board (CCB) which enforces a formal communication network between the hardware and software development groups. The Board consists of key engineers and managers from both disciplines who mutually govern the status of the system and incorporate design changes on an agreed-to basis. They attend all formal design reviews and major design walk-throughs to ensure that neither group is being compromised. They also meet on a weekly basis to review the current state of the system and examine design problem reports which are generated throughout the development effort (i.e., from requirements definition to system I&T). Finally, all design-related issues are properly documented and reviewed by the CCB before any changes are officially made to either the hardware or software designs.

TASK 2: SOFTWARE DEVELOPMENT PHASES

The second task is to divide the software project into six logical phases of development:

1) Software Requirements Definition
2) Top-Level and Detailed Design
3) Module Code and Test
4) Component I&T
5) Configuration Item I&T
6) System I&T

The first phase is to define the software requirements in relationship to the hardware requirements to ensure that design oversites are not incurred. Additionally, this phase must be completed prior to starting the second phase in order to establish a known baseline of requirements.

The second phase mandates the satisfactory completion of all of the software design reviews and walk-throughs prior to coding the software. Work folders are then established which contain all of the software material pertaining to a particular processing element called a configuration item (e. g., data processors and signal processors are two separate

DTIC COPY INSPECTED 6

configuration items). Examples of data recorded in the work folder include software requirements, top-level and detailed design descriptions, walk-through results, source listings, module code and test history, component and configuration item I&T history, and design problem reports.

The third phase requires that each software function be developed in a modular fashion and tested accordingly (i.e., a module is defined as being a small compilable file which performs a single function in an efficient manner). In addition, module development is typically performed on a mainframe computer so that the programmer can take advantage of various support tools such as the Scenario Generator/ Monitor, the ICD data base and debuggers.

Once each module meets its own unique performance requirements, then software development transitions to the component I&T phase where a set of related modules are integrated and tested as a whole entity. Again, this phase of development takes place on a mainframe to take advantage of the support tools.

The next phase is to integrate and test a set of related components which comprise a given configuration item. This development takes place on both the mainframe and the target processor so that other helpful tools may be utilized (e.g., the hardware emulator which resides on the mainframe and the stand-alone test station which interfaces to the processor). Additionally, the developer will be able to assess how well the software runs on the hardware during this phase of development.

The final phase is System I&T, whereby the configuration items are integrated together to create individual threads of operation. This is the final and most difficult stage of software development since it involves the integration of several unique hardware and software elements.

In summary, these software development phases have proven to be excellent procedures for developing software while, at the same time, enhancing program visibility and control.

TASK 3: AUTOMATED ICD DATA BASE

The use of an automated ICD data base is beneficial to the user in several respects. By its very nature, it is readily available to anyone who has the appropriate need-to-know authorization to access the host computer which supports the toolset. The data base also contains a detailed description

of every message and ICD used within the system. Therefore, written and verbal miscommunications between the various development groups are effectively minimized, if not totally eliminated.

The data base is maintained by the CCB on a weekly basis. Any proposed changes to the baselined hardware or software designs are always documented in design problem reports and formally reviewed by the CCB. If approval is granted, then the data base is updated by the CCB to reflect these modifications.

In conclusion, the automated data base is an invaluable source of design/interface data because it is readily accessible to all users and provides current information.

TASK 4: SOFTWARE TEST ENVIRONMENT

The following test environments were chosen since they verify three aspects of software design, namely the software-to-software interfaces, the hardware-to-software interfaces, and real-time operation.

Scenario generators and monitors are useful software programs which are used to stimulate the software packages while monitoring the message traffic between the software modules (i.e., software-to-software interface). Hence, the developer can easily test the software with a number of possible scenarios and examine the resultant messages.

Hardware emulators are also useful software programs which emulate the operational characteristics of the host processor. This is beneficial in determining the validity of the software algorithm in relation to the processor's capabilities and limitations (i.e., hardware-to-software interface).

Finally, the stand-alone test stations are unique hardware environments which are used to debug the software during real-time operation. This, too, is an invaluable tool since it permits the developer to examine the results from individual software instructions during actual execution times.

CONCLUSION

With the advent of highly programmable systems, there surfaced a need within the avionics community to establish guidelines which definitized the relationship between hardware and software development efforts. These guidelines evolved into four specific tasks which ensure that: 1) design oversites are minimized; 2) software progress is controlled; 3) critical interface definitions are readily available; and 4) software is fully tested prior to system I&T. It is believed, therefore, that the combined implementation of all four tasks will ensure the development of a reliable, operational system.